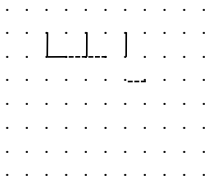
A game



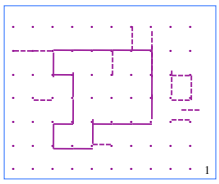
while there is room
A draws — or | ;
B draws --- or ¦ ;

A and B alternate moves

A wants to get a solid closed curve.
B wants to stop A from getting a solid closed curve.
Who can win? What strategy to use?

Board can be any size: m by n dots, with m > 0, n > 0

A won the game to the right because there is a solid closed curve.



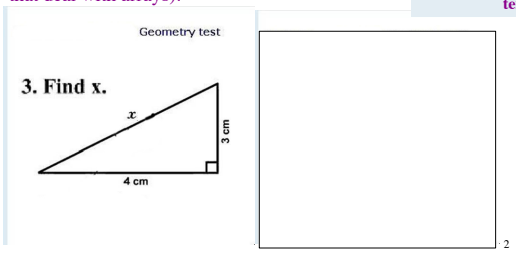
1

We develop recursive functions and look at execution of recursive functions

CS1110 6 Oct 2008
More on Recursion

Study Sect 15.1, p. 415. Watch activity 15-2.1 on the CD. In DrJava, write and test as many of the self-review exercises as you can (disregard those that deal with arrays).

Next time: Casting about. Look at Secs 4.2 and 4.3 in text



2

Executing recursive function calls.

```
/** = non-negative n, with commas every 3 digits
      e.g. commafy(5341267) = "5,341,267" */
public static String commafy(int n) {
    1: if (n < 1000) return "" + n;
    // n >= 1000
    2: return commafy(n/1000) + "," + to3(n%1000);
}

/** = p with at least 3 chars —
     0's prepended if necessary */
public static String to3(int p) {
  if (p < 10) return "00" + p;
  if (p < 100) return "0" + p;
  return "" + p;
}
```

commafy(5341266 + 1)

commafy: 1 | Demo

n

3

Recursive functions

/** = a copy of s in which s[0..1] are swapped, s[2..3] are swapped, s[3..4] are swapped, etc. */

public static String swapAdjacent(String s)

/** = b^c. Precondition: c ≥ 0*/
public static int exp(**int** b, **int** c)

Properties:

(1) $b^c = b * b^{c-1}$

(2) For c even

$b^c = (b*b)^{c/2}$

e.g 3*3*3*3*3*3*3*3

= (3*3)*(3*3)*(3*3)*(3*3)

4

Recursive functions

```
/** = b^c. Precondition: c ≥ 0*/
public static int exp(int b, int c) {
  if (c = 0)
    return 1;
  if (c is odd)
    return b * exp(b, c–1);
  // c is even and > 0
  return exp(b*b, c / 2);
}
```

c	number of calls
0	1
1	2
2	2
4	3
8	4
16	5
32	6
2^n	n + 1

32768 is 2^{15}

so b^{32768} needs only 16 calls!

Binary arithmetic

Decimal	Binary	Octal	Dec	Binary
00	00	00	$2^0 = 1$	1
01	01	01	$2^1 = 2$	10
02	10	02	$2^2 = 4$	100
03	11	03	$2^3 = 8$	1000
04	100	04	$2^4 = 16$	10000
05	101	05	$2^5 = 32$	100000
06	110	06	$2^6 = 64$	1000000
07	111	07	$2^{15} = 32768$	1000000000000000
08	1000	10		
09	1001	11		
10	1010	12		

Test c odd: Test last bit = 1

Divide c by 2: Delete the last bit

Subtract 1 when odd: Change last bit from 1 to 0.

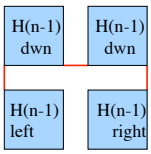
Exponentiation algorithm processes binary rep. of the exponent.

Hilbert's space-filling curve

Hilbert(1):

Hilbert(2):

Hilbert(n):



As the size of each line gets smaller and smaller, in the limit, this algorithm fills every point in space. Lines never overlap.

Hilbert's space-filling curve